

Using Temporal Logic to Specify and Verify Cryptographic Protocols (Progress Report)

James W. Gray, III¹

Department of Computer Science

Hong Kong University of Science and Technology

John McLean

Center for High Assurance Computing Systems

Naval Research Laboratory

Abstract

We use standard linear-time temporal logic to specify cryptographic protocols, model the system penetrator, and specify correctness requirements. The requirements are specified as standard safety properties, for which standard proof techniques apply. In particular, we are able to prove that the system penetrator cannot obtain a session key by any logical or algebraic techniques. We compare our work to Meadows' method. We argue that using standard temporal logic provides greater flexibility and generality, firmer foundations, easier integration with other formal methods, and greater confidence in the verification results.

1 Introduction

We have started work on a project to apply temporal logic to reason about cryptographic protocols. Some of the goals of the project are as follows.

1. Allow the user to state and prove that the penetrator cannot use logical or algebraic techniques (e.g., we are disregarding probabilistic attacks) to obtain certain words (e.g., session keys). Although this is a vital correctness condition for cryptographic protocols, as far as we know, Meadows'

¹Supported by grant HKUST 608/94E from the Hong Kong Research Grants Council.

Report Documentation Page			<i>Form Approved OMB No. 0704-0188</i>		
<p>Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p>					
1. REPORT DATE 1995	2. REPORT TYPE	3. DATES COVERED 00-00-1995 to 00-00-1995			
4. TITLE AND SUBTITLE Using Temporal Logic to Specify and Verify Cryptographic Protocols (Progress Report)			5a. CONTRACT NUMBER		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory, Center for High Assurance Computer Systems, 4555 Overlook Avenue, SW, Washington, DC, 20375			8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSOR/MONITOR'S ACRONYM(S)		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 17	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

method ([Mea92]) is the only approach that has successfully proved such statements. This is surprising in light of the large number of papers written about formal analysis of cryptographic protocols.

2. As far as possible, we will employ the standard concepts and techniques used in verifying the correctness of distributed systems. In the large number of papers on cryptographic protocol analysis, there is a general tendency to introduce new logics with many special-purpose axioms. (One exception is the work of Bieber, et. al. [BBCLvW93]; however, that work does not address our first goal.) Based on our work so far, we see no reason to introduce any new axioms. We plan to use only existing logics.

In this paper, we provide a rough idea of our approach, describe the progress to date, and outline what we believe will be the primary advantages of our approach. Since the approach of Meadows and Syverson (see [Mea92], [SM93], and [SM94]) is most closely related to our work, at various places throughout the paper we make comparisons between ours and theirs.

2 Model

An important step in any security analysis is to set out the *threats* and *capabilities* of the penetrators that are being countered. That is, we need to establish the precise objectives of the penetrator (i.e., the threats that are of concern) and the capabilities that the penetrator can use to accomplish those objectives. These objectives and capabilities must be expressible within the model that is used in our security analysis.

2.1 Threats

Roughly speaking, we want to reason about protocols that establish a key for use during a session involving two participants—the *initiating* participant, which we’ll denote by A , and the *receiving* participant, which we’ll denote by B . Typically, there is another legitimate participant of the protocol—the key server, which we’ll denote by S . The types of threats that concern us (i.e., the objectives of the penetrator) are the following. (Later, we will make these threats precise in the form of *requirements* on the protocol.)

1. During the process of distributing keys to the legitimate participants, a key is accidentally disclosed to the penetrator in addition to being accepted as a good key by one of the legitimate participants (viz, either A or B);
2. A key is accepted (by one of the legitimate participants) for a given session between participants A and B and then later accepted for a different session, perhaps with different participants.

3. One of the legitimate participants accepts a key that was *not* generated by the key server expressly for the present session;
4. B accepts a key for a session that is ostensibly with A , but in fact, was not initiated by A .

2.2 Penetrator Capabilities

Roughly speaking, we are concerned with a penetrator that has complete control over the communication network. As such, we allow the penetrator to have the following capabilities. The penetrator can:

1. intercept all outgoing messages;
2. modify any message (in any way consistent with the keys she possesses; e.g., the penetrator cannot encrypt or decrypt a piece of data with a key that she does not possess);
3. deliver any message (that she possesses) to any legitimate participant;
4. start up any number of legitimate participants (e.g., if it furthers her objectives, the penetrator could start up multiple instances of a participant acting as the initiator of the protocol.).

2.3 Justification of the Penetrator's Capabilities

It may seem that we are giving the penetrator too much power. For example, in the above, we have given the penetrator *complete* control over the network. It is as if every aspect of the network (barring the relatively few “legitimate” participants of the protocol, viz, A , B , and S) has been subverted by the penetrator. However, allowing the penetrator to have such control essentially gives us a *worst-case* analysis of the security of the protocol. That is, when reasoning about security, we typically want to know what is the worst possible situation that the system can get in. Therefore, we have given the penetrator the greatest possible capability short of subverting the legitimate participants of the protocol.

In addition to these considerations, one of the capabilities described in the previous section merits some further justification. Namely, the capability to “start up any number of legitimate participants” is provided so that we can reason about arbitrary interleaving attacks, such as those described in [DvOW92].

2.4 Definition of the Model

The above threats and capabilities motivate the model we adopt, which was originally formulated by Dolev and Yao [DY83] and later substantially generalized by Meadows [Mea92]. In this section, we set out this model.

We start by describing our general model of distributed computation, which is based on a standard model from the distributed computing literature, namely, the so-called “interleaving” model in which all events—including concurrent events—occurring during an execution of the system are interleaved to form a single “trace” of that execution (see, e.g., [Lam91]). As is standard, we will consider a *trace* to be an infinite sequence of system states.² The system is then characterized by its set of possible traces.

The distributed system consists of a set of agents, $P_1, P_2, \dots, P_i, \dots$, each with a local state, along with a communication medium. A global system state (i.e., an element of a system trace) is composed of the local state for each agent together with the state of the communication medium.

Each agent’s local state is represented by a set of local state variables. We will indicate that a particular state variable, say x , is local to a particular agent, say P_i , using a dot notation similar to the notation used for references to fields of a record in Pascal, viz., “ $P_i.x$ ”. For example, each agent may use a local program counter named pc to keep track of its local execution of the protocol. Program counters local to different agents can be distinguished by the dot notation.

We represent the communication medium as a set of messages, μ . We will write $\text{send}(P_i, P_j, m)$ to denote that agent P_i sends the message m (i.e., places it on the communication medium) addressed to P_j . We make this precise using Lamport’s Raw Temporal Logic of Actions (RTLA) [Lam91].³

Recall that in RTLA, an action is a statement about pairs of states. Unprimed state variables (e.g., μ) refer to the value in the first state and primed state variables (e.g., μ') refer to the value in the second state. We define the send action as follows.

$$\text{send}(P_i, P_j, m) \iff \mu' = \mu \cup \{(P_i, P_j, m)\}$$

Similarly, we write $\text{receive}(P_i, P_j, x)$ to denote that P_i blocks until there is a message on the communication medium from P_j and then the contents of that message are copied to P_i ’s local state variable, x . We make this precise using

²A “terminating” computation is modeled as an infinite trace that has a finite prefix representing the computation prior to “termination”, the final state of which is repeated (or “stuttered”) infinitely to form the remainder of the trace [Lam91].

³We use *Raw TLA* since in the present paper we are unconcerned with the issues of refinement.

RTLA as follows.

$$\begin{aligned} \text{receive}(P_i, P_j, x) \iff & (\exists m) [(P_j, P_i, m) \in \mu \quad \wedge \\ & \mu' = \mu - \{ (P_j, P_i, m) \} \quad \wedge \\ & P_i.x' = m] \end{aligned} \quad (1)$$

We also allow agents to receive messages without specifying the sender. We denote such an action by $\text{receive}(P_i, p, x)$, where p is a state variable local to P_i . We define this action as follows.

$$\begin{aligned} \text{receive}(P_i, p, x) \iff & (\exists P_j) [(P_j, P_i, m) \in \mu \quad \wedge \\ & \mu' = \mu - \{ (P_j, P_i, m) \} \quad \wedge \\ & P_i.x' = m \quad \wedge \\ & P_i.p' = P_j] \end{aligned} \quad (2)$$

Now, the capabilities of the penetrator can be expressed in terms of the model as follows. We will have a designated agent, E , representing the penetrator. Part of the penetrator's local state is a set of "words" that she possesses, denoted $E.\text{words}$. This set of words includes the messages that she has received over the communication medium as well as words the agent has computed by performing operations (e.g., encryption) over the set of words she already possesses.

E nondeterministically performs some sequence of actions, chosen from the following possibilities.

- intercept a message and add it to $E.\text{words}$;
- produce a new word by performing some operation (e.g., encryption) on some words that she already possesses and add the new word to $E.\text{words}$. Note that if the penetrator already possesses some encrypted word $\text{encrypt}(m, k)$, she must also possess the encryption key, k , in order to produce the clear-text message, m .
- send some word that is in $E.\text{words}$ to any legitimate participant (ostensibly from any other participant).

Before formalizing these actions, we recall a particular action that is often useful in RTLA specifications; namely, $\text{Unchanged}(\alpha)$, where α is a set of state variables, specifies that every variable in α does not change. More precisely, we'll use the following definition.⁴

$$\text{Unchanged}(\{v_1, v_2, \dots, v_n\}) \iff v_1 = v'_1 \wedge v_2 = v'_2 \wedge \dots \wedge v_n = v'_n$$

For convenience in specifying the set of variables that do not change, let σ be the set of *all* state variables, including all agents' local state and the communication medium. We define the action of the penetrator intercepting a message by

⁴This definition of $\text{Unchanged}(\alpha)$ is actually a special case of Lamport's definition, but it is sufficient for our purposes.

$$\begin{aligned} \text{intercept} \iff & (\exists P_i, P_j, m)[(P_i, P_j, m) \in \mu \quad \wedge \\ & \mu' = \mu - \{ (P_i, P_j, m) \} \quad \wedge \\ & E.\text{words}' = E.\text{words} \cup \{m\} \quad \wedge \\ & \text{Unchanged}(\sigma - \{\mu, E.\text{words}\}) \quad] \end{aligned}$$

The action of the penetrator producing a new word from other words she already possesses is defined as follows.

$$\begin{aligned} M1 \iff & (\exists m, K)[m \in E.\text{words} \wedge K \in E.\text{words} \wedge \\ & E.\text{words}' = E.\text{words} \cup \{ \text{encrypt}(m, K) \}] \\ & \wedge \text{Unchanged}(\sigma - \{E.\text{words}\}) \end{aligned}$$

$$\begin{aligned} M2 \iff & (\exists m, K)[m \in E.\text{words} \wedge K \in E.\text{words} \wedge \\ & E.\text{words}' = E.\text{words} \cup \{ \text{decrypt}(m, K) \}] \\ & \wedge \text{Unchanged}(\sigma - \{E.\text{words}\}) \end{aligned}$$

$$\begin{aligned} M3 \iff & (\exists m, K)[\text{decrypt}(\text{encrypt}(m, K), K) \in E.\text{words} \wedge \\ & E.\text{words}' = E.\text{words} \cup \{m\}] \\ & \wedge \text{Unchanged}(\sigma - \{E.\text{words}\}) \end{aligned}$$

$$\begin{aligned} M4 \iff & (\exists m, K)[\text{encrypt}(\text{decrypt}(m, K), K) \in E.\text{words} \wedge \\ & E.\text{words}' = E.\text{words} \cup \{m\}] \\ & \wedge \text{Unchanged}(\sigma - \{E.\text{words}\}) \end{aligned}$$

$$\text{manipulate} \iff M1 \vee M2 \vee M3 \vee M4$$

The action of the penetrator sending a message to one legitimate participant ostensibly from another legitimate participant is defined by

$$\begin{aligned} \text{impersonate} \iff & (\exists P_i, P_j, m)[m \in E.\text{words} \quad \wedge \\ & \text{send}(P_i, P_j, m) \quad \wedge \\ & \text{Unchanged}(\sigma - \{\mu\})] \end{aligned}$$

We can now define a penetrator action as consisting of one of the above actions.

$$E\text{-action} \iff \text{intercept} \vee \text{manipulate} \vee \text{impersonate} \quad (3)$$

The capability of starting up any number of legitimate participants will be modeled by placing an infinite number of initiators $\{A_1, A_2, \dots\}$, an infinite number of receivers $\{B_1, B_2, \dots\}$, and (if appropriate) an infinite number of key servers $\{S_1, S_2, \dots\}$ in parallel with the penetrator, E . That is, the A_i , B_j , and S_k will be among the set of agents, P_1, P_2, \dots . Then, the penetrator can make use of as many of these legitimate participants as needed to launch an attack.

2.5 The Flexibility of the Model

A few comments on the flexibility of our model of the penetrator are in order.

First, note that the $M3$ and $M4$ actions are our formalization of the algebraic properties of the encryption algorithm used in the protocol. To analyze a protocol that uses an encryption algorithm with different algebraic properties, we can simply substitute actions describing those properties for $M3$ and $M4$. For example, if a protocol uses the “exclusive or” (\oplus) operator, we might want to model the commutativity property of \oplus by including the following action.

$$\begin{aligned} M3' \iff & (\exists w_1, w_2) [\oplus(w_1, w_2) \in E.\text{words} \wedge \\ & E.\text{words}' = E.\text{words} \cup \{\oplus(w_1, w_2)\}] \\ & \wedge \text{Unchanged}(\sigma - \{E.\text{words}\}) \end{aligned}$$

In contrast, making such a change in Meadows’ method would present serious difficulties. This is because Meadows models the algebraic properties of the encryption algorithm as a Noetherian and locally confluent term rewriting system. In such systems, all terms have a unique canonical form—the so-called “reduced form”. However, due to commutativity, $\oplus(w_1, w_2)$ can be rewritten as $\oplus(w_2, w_1)$ and then back again; there is no unique canonical form. Accommodating such an encryption algorithm would require a major change to her tool. This is because the algorithm at the heart of her tool—the narrowing algorithm—depends on the Noetherian and locally confluent properties. At each step of her analysis, all words are rewritten in reduced form.

Second, note that we could easily insert additional penetrator capabilities into Formula 3. For example, if we wanted to reason about attacks such as the Denning-Sacco attack on the Needham-Schroeder protocol [DS81], we could include a “compromise key” capability, which could be used by the penetrator to obtain previously used keys.

The possibilities of adding penetrator capabilities and changing the underlying encryption algorithm illustrate the flexibility and power of modelling the penetrator in temporal logic. Further, our approach makes the model of the penetrator explicit. As we will see in the next section, we use the same language to specify the protocol being analyzed and the requirements being proven. Thus, all definitions and assumptions used in the analysis are stated in a single language, rather than, e.g., being partly buried in the definition of the automated tool.

3 A First Example

As our first example, we chose something extremely simple. Our motivations are to provide a rough idea of what's involved in applying our approach and to provide some evidence that the approach is feasible.

The example we chose cannot even be called a protocol; it is the example used by Meadows in [Mee92, pages 15–16] to illustrate the use of her method. In Meadows' formalism, the example consists of a single rule, viz,

$$\text{If } Y \subseteq \mathbf{W} \text{ and } \text{KEYSTATE}(Z) = X \text{ then } \mathbf{W} := \mathbf{W} \cup \text{encrypt}(Y, X)$$

where \mathbf{W} is the set of words known to the penetrator, which we've called $E.\text{words}$.

The intuition behind this rule is that the legitimate protocol participant, Z , possesses a key, denoted $\text{KEYSTATE}(Z)$. Whenever Z receives a word, it encrypts that word with $\text{KEYSTATE}(Z)$ and returns the result. We can specify this rule in our approach as follows.

$$\begin{aligned} Z1 \iff & Z.pc = 1 & \wedge \\ & \text{receive}(Z, Z.p, Z.x) & \wedge \\ & Z.pc' = 2 & \wedge \\ & \text{Unchanged}(\sigma - \{\mu, Z.p, Z.x, Z.pc\}) & \\ \\ Z2 \iff & Z.pc = 2 & \wedge \\ & \text{send}(Z, Z.p, \text{encrypt}(Z.x, \text{KEYSTATE}(Z))) & \wedge \\ & Z.pc' = 1 & \wedge \\ & \text{Unchanged}(\sigma - \{\mu, Z.pc\}) & \\ \\ Z\text{-Init} \iff & Z.pc = 1 \wedge Z.x = \text{NULL} & \end{aligned}$$

We specify the protocol running in parallel with the penetrator as

$$R \iff Z\text{-Init} \wedge \mu = \{\} \wedge E\text{-Init} \wedge \square(Z1 \vee Z2 \vee E\text{-Action}) \quad (4)$$

(where $E\text{-Init}$ will be described below).

Our description is clearly not as compact as Meadows'. There are two reasons for this.

1. We have included a program counter pc that allows Z to keep track of its place in the protocol. In the present example, it simply alternates between receiving and sending. In Meadows' formalism, the act of receiving a message and sending a reply is treated as a single action and so she does not need to maintain a program counter for this simple example. In more complex examples, involving multiple rules that are performed in sequence, Meadows' formalization would be similar to ours.

2. We state implicitly which variables do not change. In Meadows' method, if a variable is not mentioned, it is assumed to remain unchanged. We could adopt this approach too. However, as pointed out by Lamport [Lam91, page 59] this introduces an “inherent complexity epitomized by the observation that $y' = y'$ is not equivalent to **true**.” (The former allows y to change, whereas the latter does not.)

For this first example, we want to prove that the penetrator cannot obtain a particular word, a . This is easily formalized in temporal logic as

$$R \rightarrow \square(a \notin E.\text{words})$$

(where R is the specification of the protocol). The standard approach to proving such a formula is to prove that

$$a \notin E.\text{words} \tag{5}$$

is an invariant over the system transition relation. However, as is typically the case, this formula is not strong enough to be directly proved invariant. Meadows describes why this is the case in terms of her tool [Mea92, page 16]. In terms of temporal logic, the problem is as follows.

Suppose we have a state where the penetrator has obtained $\text{encrypt}(a, k)$ and k for some word k , but not yet obtained a . In this state, Equation 5 is true. But, in a single transition, the penetrator can obtain a (by performing the decryption) and Equation 5 will be true. Thus, Equation 5 is not a sufficient condition on the pre-state to ensure that Equation 5 will hold in the post-state.

Meadows solves this problem by defining a formal language by way of the following context-free grammar.⁵

$$\begin{aligned} \mathbf{K} &\rightarrow a \\ \mathbf{K} &\rightarrow \text{encrypt}(\mathbf{K}, \mathbf{L}) \\ \mathbf{K} &\rightarrow \text{decrypt}(\mathbf{K}, \mathbf{L}) \end{aligned}$$

where \mathbf{L} is the language consisting of all words not containing variables. Meadows then proves that

The penetrator does not possess any word in \mathbf{K} .

is invariant. The technique she uses to prove this is rather complicated and only partly formalized. It involves transforming the grammar rules into prolog goals, running her automated tool on them, and then interpreting the results again in terms of the context-free grammar.

⁵We've made a few minor syntactic changes to Meadows' grammar to bring it in line with our notation.

In contrast, we can formalize the entire proof in terms of temporal logic. First, we specify the set of words \mathbf{K} as follows.

$$(\forall w)w \in \mathbf{K} \iff \left(\begin{array}{l} w = a \\ (\exists k, l)[w = \text{encrypt}(k, l) \wedge k \in \mathbf{K}] \\ (\exists k, l)[w = \text{decrypt}(k, l) \wedge k \in \mathbf{K}] \end{array} \right) \vee \quad (6)$$

Since we explicitly include the communication medium μ and the participant's local variable $Z.x$ in our model, we cannot simply use

$$(\forall w)[w \notin E.\text{words} \vee w \notin \mathbf{K}]$$

as our invariant. In particular, if a word in \mathbf{K} is present on the communication medium μ or in the variable $Z.x$ (when Z 's program counter is 2), then it can be obtained by the penetrator. Therefore, we use the following formula as our invariant.

$$\text{INV} \iff \left(\begin{array}{l} (Z.x \notin \mathbf{K} \vee Z.pc \neq 2) \\ (\forall w)[w \notin E.\text{words} \vee w \notin \mathbf{K}] \\ (\forall P_1, P_2, w)[(P_1, P_2, w) \notin \mu \vee w \notin \mathbf{K}] \end{array} \right) \wedge \quad (7)$$

Now we can see what is needed as the penetrator's initial condition, namely, we need an initial condition that is strong enough that we will be able to prove INV is initially true. The following is sufficient.

$$E\text{-Init} \iff (\forall w)[w \notin E.\text{words} \vee w \notin \mathbf{K}]$$

We can now state and prove our theorem.

Theorem 3.1

$$R \Rightarrow \square(a \notin E.\text{words})$$

Proof: In overview, the proof goes like this. We use induction to prove

$$R \Rightarrow \square(\text{INV}) \quad (8)$$

and we show that

$$\text{INV} \Rightarrow a \notin E.\text{words} \quad (9)$$

and finally, from Formulas 8 and 9 we can easily establish the theorem.

Proving Formulas 8 and 9 are straightforward, mainly requiring a lot of case checking. To provide an idea of what's involved, we prove part of Formula 8.

From the definition of R (Formula 4), we see that it is sufficient to prove a base case:

$$(Z\text{-}Init \wedge \mu = \{\} \wedge E\text{-}Init) \Rightarrow \text{INV} \quad (10)$$

and an induction case:

$$((Z1 \vee Z2 \vee E\text{-}Action) \wedge \text{INV}) \Rightarrow \text{INV}' \quad (11)$$

(where INV' is the formula obtained from INV by replacing all program variables with their primed counterparts).

We prove Formula 10 by assuming $Z\text{-}Init$, $\mu = \{\}$, $E\text{-}Init$, and $\neg\text{INV}$ and showing that these assumptions lead to a contradiction. Applying some definitions (and distributing the \neg as far as possible), we have the following assumptions.

- (a1) $Z.pc = 1$
- (a2) $Z.x = \text{NULL}$
- (a3) $\mu = \{\}$
- (a4) $(\forall w)[w \notin E.\text{words} \vee w \notin \mathbf{K}]$
- (a5) $(Z.x \in \mathbf{K} \wedge Z.pc = 2) \vee$
 $(\exists w)[w \in E.\text{words} \wedge w \in \mathbf{K}] \vee$
 $(\exists P_1, P_2, w)[(P_1, P_2, w) \in \mu \wedge w \in \mathbf{K}]$

We break (a5) into three cases corresponding to the three disjuncts.

Case 1: $Z.x \in \mathbf{K} \wedge Z.pc = 2$

From (a1), we immediately obtain the contradiction $2 = 1$.

Case 2: $(\exists w)[w \in E.\text{words} \wedge w \in \mathbf{K}]$

Existentially instantiating w (to a constant $w0$), we have

$$w0 \in E.\text{words} \wedge w0 \in \mathbf{K}$$

and instantiating w in (a4) to $w0$, we obtain a contradiction.

Case 3: $(\exists P_1, P_2, w)[(P_1, P_2, w) \in \mu \wedge w \in \mathbf{K}]$

Existentially instantiating P_1 , P_2 , and w to constants and making use of (a3), we have $(P_1, P_2, w) \in \{\}$, which is a contradiction of basic set theory.

Thus, in all cases we have a contradiction and Formula 10 is proved.

To prove Formula 11 we again proceed by contradiction. We assume $(Z1 \vee Z2 \vee E\text{-}Action)$ and INV and $\neg\text{INV}'$. Applying some definitions, we have the following assumptions.

- (b1) $Z1 \vee Z2 \vee \text{intercept} \vee \text{impersonate} \vee M1 \vee M2 \vee M3 \vee M4$
- (b2) $Z.x \notin \mathbf{K}$
- (b3) $(\forall w)[w \notin E.\text{words} \vee w \notin \mathbf{K}]$
- (b4) $(\forall P_1, P_2, w)[(P_1, P_2, w) \notin \mu \vee w \notin \mathbf{K}]$
- (b5) $(Z.x' \in \mathbf{K} \wedge Z.pc' = 2) \vee (\exists w)[w \in E.\text{words}' \wedge w \in \mathbf{K}] \vee (\exists P_1, P_2, w)[(P_1, P_2, w) \in \mu' \wedge w \in \mathbf{K}]$

We can now proceed by considering a large number of cases. In particular, there are eight cases corresponding to the eight disjuncts of (b1) and each of those has three subcases corresponding to the three disjuncts of (b5). Hence, there are a total of 24 subcases. Since all of these subcases are straightforward, we give only a few as a sample of the kind of reasoning that is involved.

Case 1: $Z1$

By the definition of $Z1$, we have the following.

- (c1.1) $Z.pc = 1$
- (c1.2) $\text{receive}(Z, Z.p, Z.x)$
- (c1.3) $Z.pc' = 2$
- (c1.4) $\text{Unchanged}(\sigma - \{\mu, Z.p, Z.x, Z.pc\})$

Subcase 1.1: $Z.x' \in \mathbf{K} \wedge Z.pc' = 2$

By (c1.2), the definition of receive (Formula 2), and existential instantiation, we have $(P_j, Z, m) \in \mu$ (where P_j is a constant) and $Z.x' = m$. By the Subcase 1.1 assumption, we have

$$(P_j, Z, m) \in \mu \wedge m \in \mathbf{K}$$

and instantiating (b4) with $P_1 = P_j$, $P_2 = Z$, and $w = m$, we obtain a contradiction.

Subcase 1.2: $(\exists w)[w \in E.\text{words}' \wedge w \in \mathbf{K}]$

By existential instantiation, we have $w0 \in E.\text{words}'$ and $w0 \in \mathbf{K}$ for some constant $w0$. By (c1.4) we know that $E.\text{words}' = E.\text{words}$. Therefore, we have that

$$w0 \in E.\text{words} \wedge w0 \in \mathbf{K}$$

which, by appropriately instantiating (b3), is a contradiction.

Subcase 1.3: $(\exists P_1, P_2, w)[(P_1, P_2, w) \in \mu' \wedge w \in \mathbf{K}]$

By existential instantiation, we have $(P_1, P_2, w0) \in \mu'$ and $w0 \in \mathbf{K}$ for constants P_1 , P_2 , and $w0$. Further, by (c1.2), the definition of receive (Formula 2), and existential instantiation, we have $\mu' = \mu - \{(P_j, Z, m)\}$ (where P_j and m are constants). Thus, we have $(P_1, P_2, w0) \in \mu - \{(P_j, Z, m)\}$, which, by basic set theory, implies

$$(P_1, P_2, w0) \in \mu \wedge w0 \in \mathbf{K}$$

which, by appropriately instantiating (b4), is a contradiction.

Cases 2–8: These cases are similar in complexity to Case 1.

□

From the above proof we see that given a usable invariant (INV) proving that a penetrator cannot obtain a key reduces to straightforward logic. We make two remarks.

1. We do not need fancy epistemic, nonmonotonic, or other special purpose logics. Properties of cryptographic protocols can be proved using the standard approach used for other properties in distributed systems.
2. There is a lot of room for automation. The proof techniques involved are not sophisticated and we expect that standard theorem provers can be successfully applied to cryptographic protocols.

4 Other Protocol Requirements

As mentioned previously, it is essential that we are able to prove that the penetrator cannot obtain particular words. However, there are many more properties that we will want to specify and prove. For example, we can adapt the requirements described by Syverson and Meadows [SM94] to our model. Our formalization of these requirements will be rather different from theirs. We discuss the differences below. Now we describe one of their requirements, both informally and formally.

No Key Reuse (informal statement) Once a key has been accepted for a given session between two given participants, it will never be accepted again for a different session or with different participants.

We will formalize this requirement by first assuming that each legitimate participant has a local state variable called *accept*. We think of this variable as a four dimensional array indexed by a key, a session ID, an initiating principal, and a receiving principal. We will denote particular elements of *accept* as $\text{accept}(K, M, A, B)$.

Each legitimate participant of the protocol will manipulate its *accept* array as follows.

- Initially, each principal's entire array will be initialized to **false**.
- Whenever a principal accepts K as the key for a given session, M , with A as the initiator and B as the receiver, that principal will set $\text{accept}(K, M, A, B)$ to **true**. (Note: for any legitimate participant in the protocol, P , the third index of P 's *accept* array will represent the initiator and the fourth index will represent the receiver; this is independent of whether P is the initiating or receiving principal.)
- Once an element of the *accept* array has been set to **true**, it will never be reset to **false**.

Thus, for a given principal, P , P 's *accept* array is meant to indicate which keys P has accepted (at any time in the past) for which sessions involving which principals.

We remark that the *accept* array is not meant to be a part of the actual protocol implementation. It is better thought of as an auxiliary variable, analogous to, e.g., “history variables” used in standard proofs of program correctness (e.g., see [AL91]). Its primary purpose is to make the specification and proof of our requirements easier. In an actual implementation, it would be omitted or, at the very least, its implementation would be improved to save on storage space.

Given the above usage of the *accept* array, we can formalize the above requirement as follows.

$$\begin{aligned} \square((P_1.\text{accept}(K, M_1, A_1, B_1) \wedge P_2.\text{accept}(K, M_2, A_2, B_2)) \\ \Rightarrow (M_1 = M_2 \wedge A_1 = A_2 \wedge B_1 = B_2)) \end{aligned} \quad (12)$$

(where P_1 , P_2 , K , M_1 , M_2 , A_1 , A_2 , B_1 , and B_2 are all universally quantified variables.)

Note that this is a standard safety property. In particular, Equation 12 is a straightforward *state invariant* and we believe we can again use the standard approach to verify it.

Now we discuss the difference between our formalization and Syverson and Meadows'. They formalized the “No Key Reuse” requirement as follows.

$$\begin{aligned} \text{accept_init}(\text{user}(A, \text{honest}), \text{user}(B, \text{honest}), K, M1) \rightarrow \\ \neg(\Diamond \text{accept_init}(\text{user}(C, \text{honest}), \text{user}(D, X), K, M2)) \quad \wedge \\ (\Diamond \text{accept_rec}(\text{user}(C, \text{honest}), \text{user}(D, X), K, M3) \rightarrow (C = B \wedge D = A)) \end{aligned}$$

(where \Diamond means “at some time in the past” and all variables are universally quantified).

One difference between our formalization and the above is that Syverson and Meadows use *actions* to indicate that a particular participant has accepted a key, whereas we have used state variables. For example, `accept_init` is an *action* indicating that a given participant, acting in the initiating role, accepts a given key for a given session. We do not believe that this difference is particularly significant in itself; the two approaches to identifying this event may be interchangeable.

A more significant difference is that in Syverson and Meadows' language, there is an implicit quantification over all time. That is, we can think of the above formula as having the “always” operator (\Box) out front. Thus, we see that their formalization makes use of nested temporal operators. In their approach, proving such a formula involves checking numerous paths through the set of possible executions.

Essentially, Syverson and Meadows are making use of simple temporal reasoning. We therefore believe that we will be able to complete an analogous proof using temporal logic (although we haven't tried it yet). However, we also believe we will be able to complete a simpler proof by using our invariant. In particular, our invariant is free of temporal quantifiers and so we should be able to complete the proof by simply reasoning about state pairs; no temporal reasoning will be required. Of course, it remains to be seen whether this will work out as we expect.

5 Further Comparison to Meadows' Method

In Meadows' method, things are rather disjoint.

1. The model of the penetrator is partly specified as a few prolog rules (i.e., the set of Noetherian and locally confluent rewrite rules) and partly integrated into the tool (i.e., manifesting itself in the form of the language for specifying the protocol).
2. Context-free grammars are used to specify sets of words that are unobtainable by the penetrator.

3. Requirements are specified in temporal logic and then translated by hand into prolog goals [SM93][SM94].
4. The protocol is specified as a set of prolog rules.
5. Due to the variety of languages used to describe various aspects of theorems, the proofs in Meadows' method are necessarily carried out using a mixture of formal and informal techniques.

In contrast, in our method, all of the above are carried out using temporal logic. The protocol, the correctness requirements of the protocol, the capabilities of the penetrator, and all ancillary definitions such as sets of unobtainable words are specified as formulas in temporal logic. Further, all proofs are carried out in temporal logic.

6 Discussion

We have shown that standard linear-time temporal logic can be used to specify cryptographic protocols, model a system penetrator, and specify correctness requirements. Although we have yet to apply our technique to a wide variety of protocol properties, we believe it is clear that the modeling environment presented in this paper is sufficient to reason about standard protocol properties. By not being specific to cryptographic protocols *per se*, it also affords a wide degree of flexibility not available in other techniques. For example, although we have given a penetrator only the ability to intercept a message, manipulate his current word set, and impersonate a protocol participant, we could easily add further abilities (e.g., to compromise a key). Other advantages we expect to achieve with our method include the following.

1. Our approach is more general than Meadows' work. As discussed in Section 2.5, we are not limited by the particular algebraic properties of the underlying encryption algorithm.
2. Our approach will be easily integrated into other formal methods. Since we use standard concepts from temporal logic, it should be straightforward to integrate proofs of other properties (e.g., fault tolerance properties) with our proofs. We even expect to be able to use the same protocol specification and prove that it satisfies security requirements as well as other properties. Further, it should be possible to make use of general purpose theorem provers to carry out the verification.
3. Our method will provide greater assurance in the correctness of the protocol than Meadows' method. This is because the result of applying our method will be a clear-cut theorem and proof. The theorem will state that

a particular protocol, executing in parallel with a particular penetrator, satisfies a particular goal, where all definitions and assumptions are stated as formulas in a single logic. Further, the proof will be carried out in that same logic.

References

- [AL91] Martín Abadi and Leslie Lamport. On the existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.
- [BBCLvW93] Pierre Bieber, Nora Boulahia-Cuppens, Thomas Lehmann, and Erich van Wickeren. Abstract machines for communication security. In *Proc. Computer Security Foundations Workshop VI*, Franconia, NH, June 1993. IEEE Computer Society Press.
- [DS81] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.
- [DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2(2):107–125, June 1992.
- [DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.
- [Lam91] Leslie Lamport. The temporal logic of actions. Technical Report 79, DEC Systems Research Center, Palo Alto, CA, December 1991.
- [Mea92] Catherine Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–35, 1992.
- [SM93] Paul Syverson and Catherine Meadows. A logical language for specifying cryptographic protocol requirements. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 165–177, Oakland, CA, May 1993.
- [SM94] Paul Syverson and Catherine Meadows. Formal requirements for key distribution protocols. In *Pre-proceedings of EUROCRYPT ‘94*, pages 325–337, University of Perugia, Italy, May 1994. The proceedings of *EUROCRYPT ‘94* will be published in the Springer-Verlag Lecture Notes in Computer Science series.